Introducing Robin

An Interactive Modeling Assistant

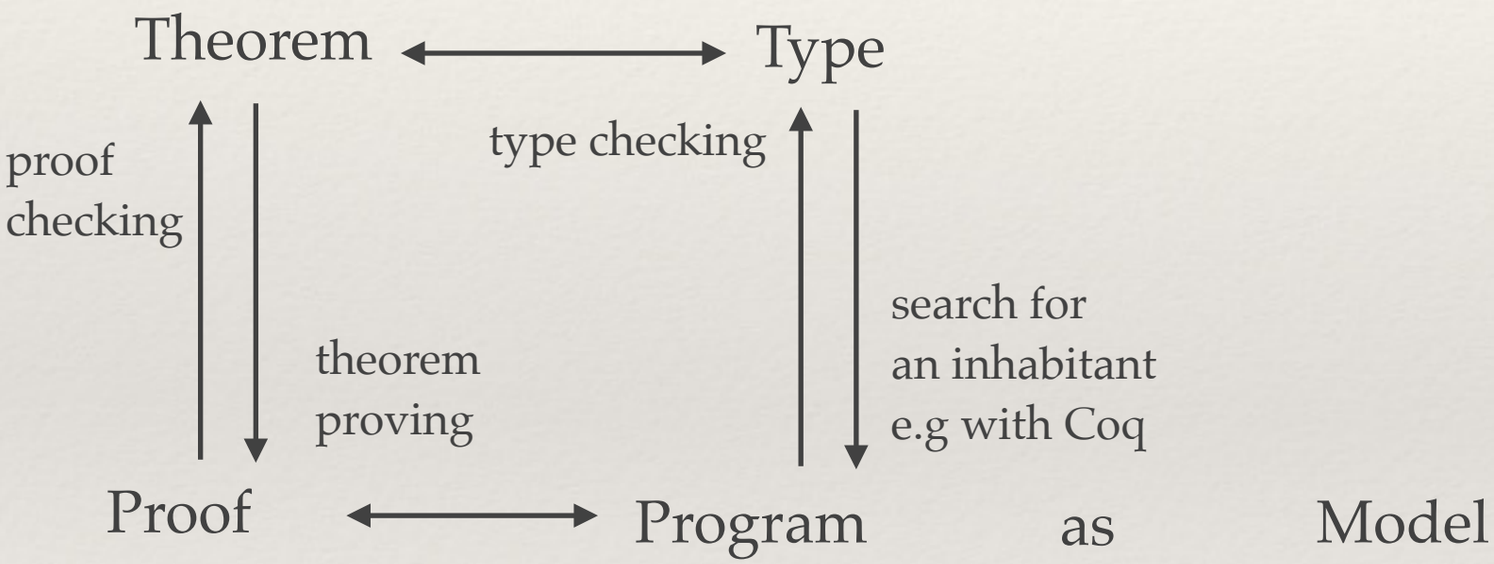
Adrien Husson and Jean Krivine - PPS

Proofs as models

Program as Model

Proofs as models

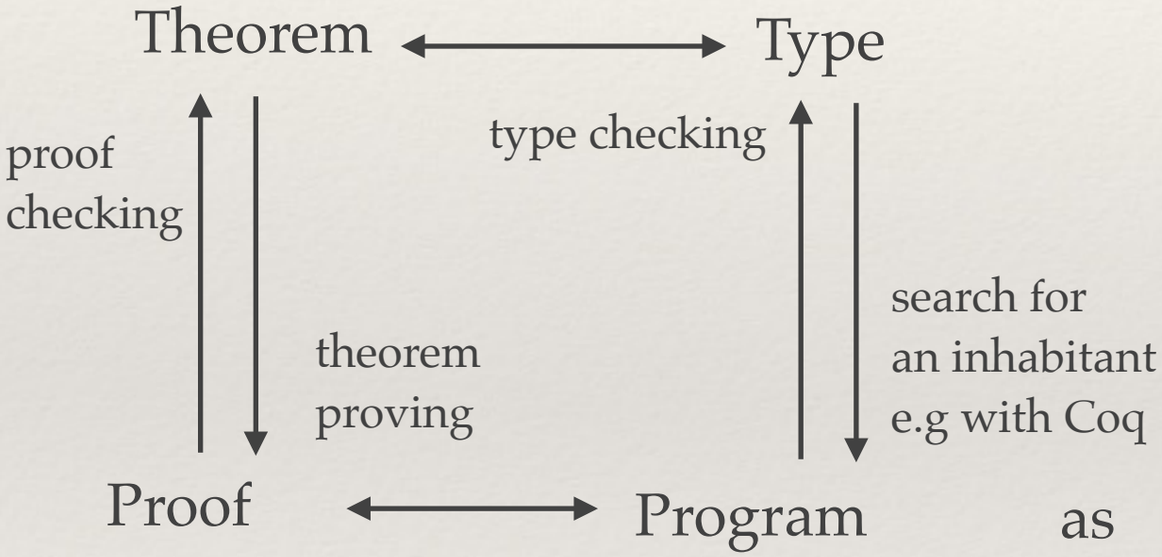
When is a proof complete?



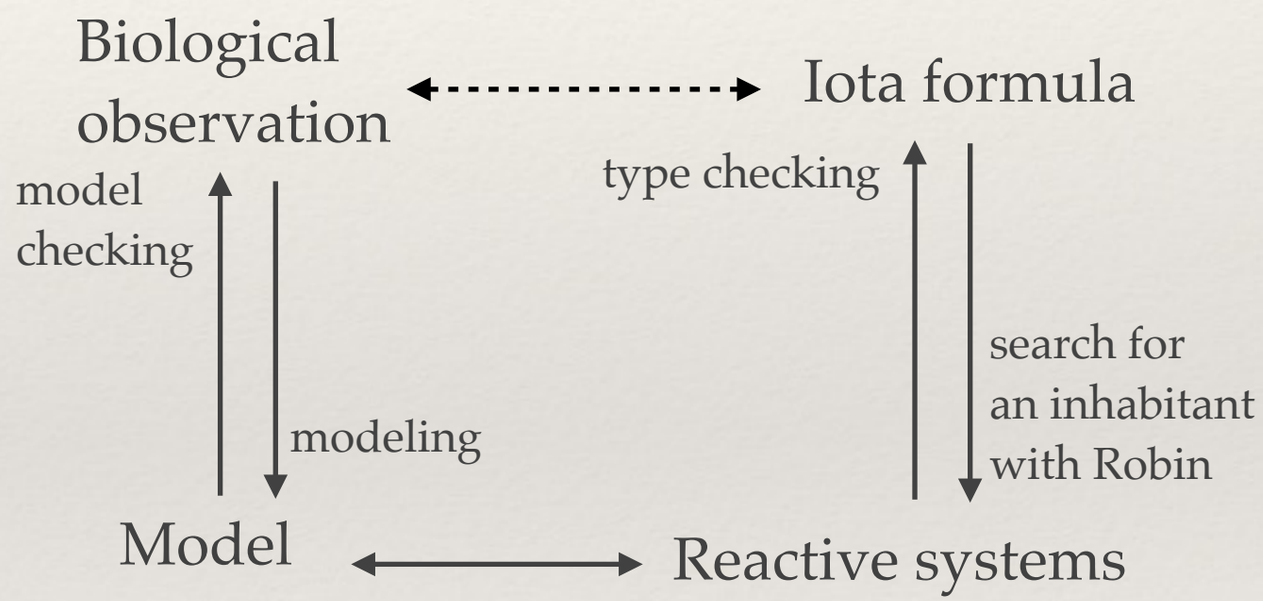
Curry-Howard correspondance

Proofs as models

When is a proof complete?



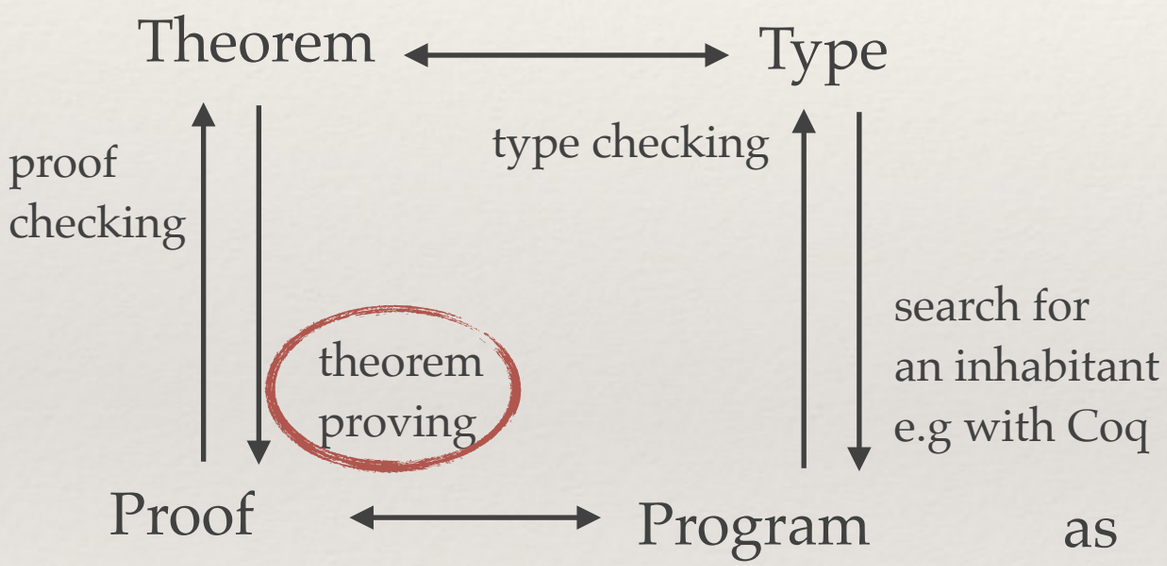
When is a model complete?



Curry-Howard correspondance

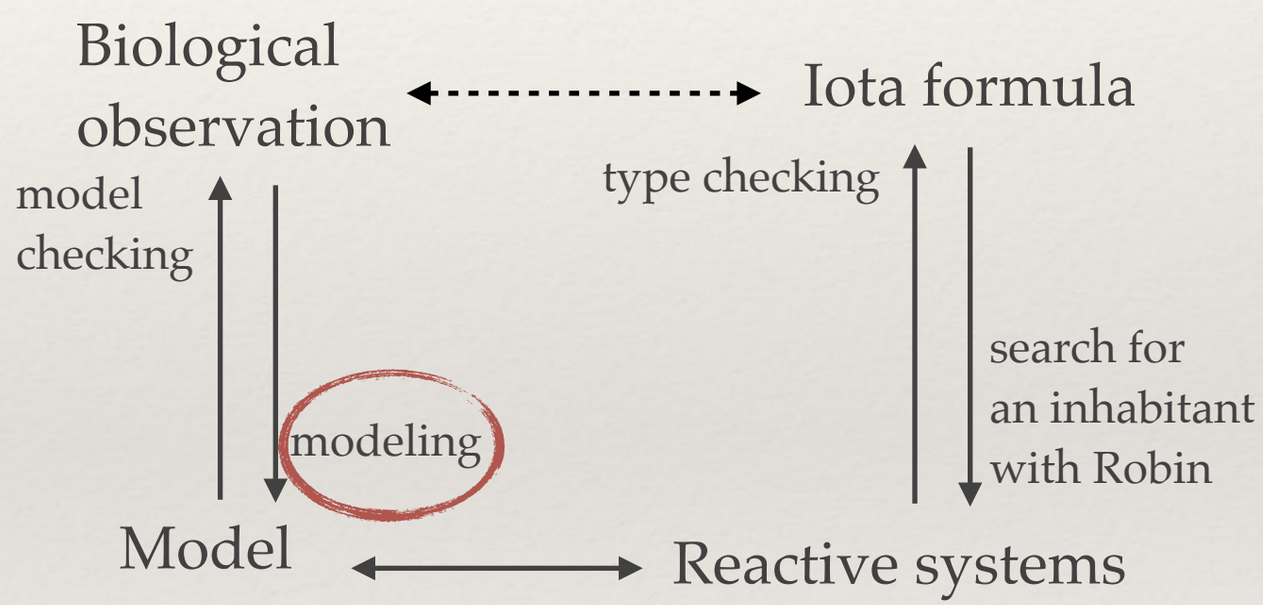
Proofs as models

When is a proof complete?



Curry-Howard correspondance

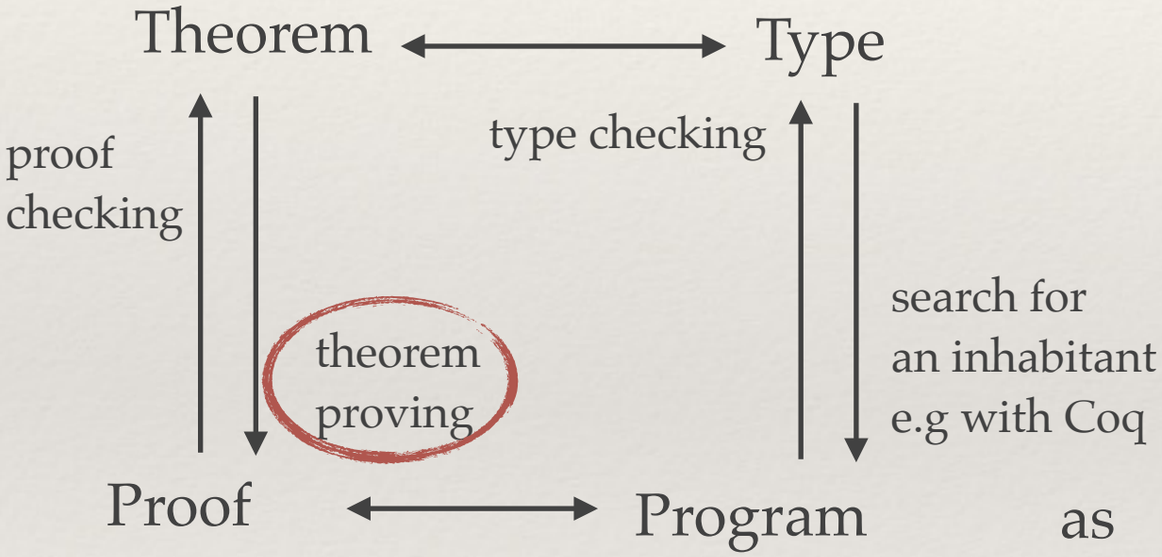
When is a model complete?



Hard (and informal) task one tries to mechanize

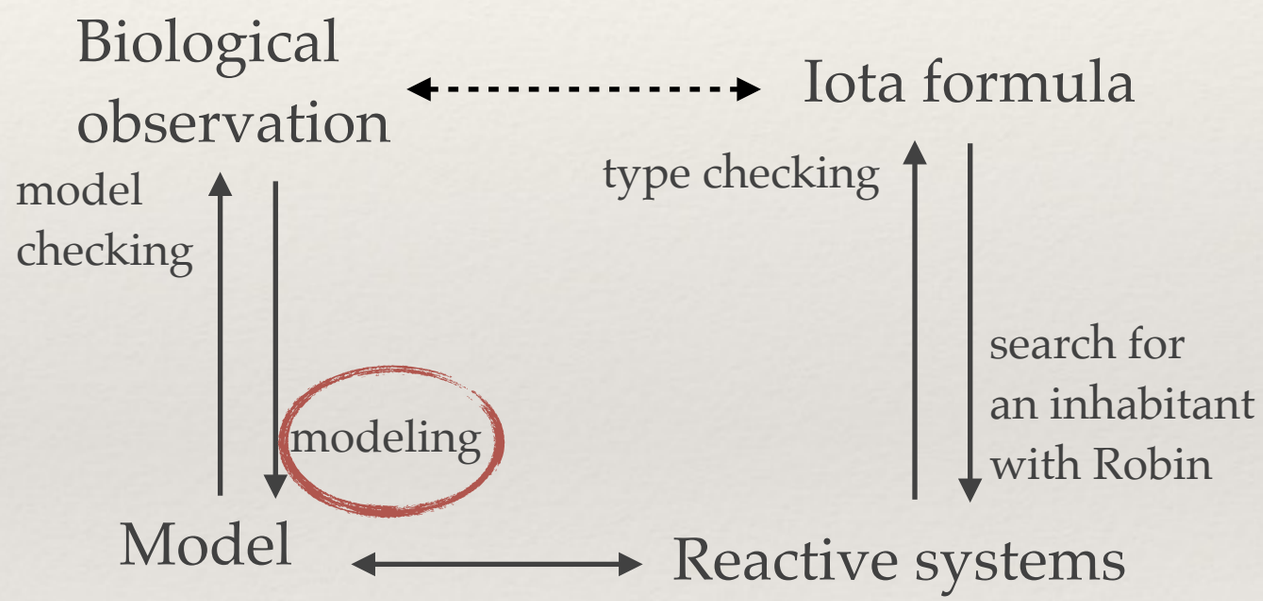
Proofs as models

When is a proof complete?



Curry-Howard correspondance

When is a model complete?



Krivine-Harvard correspondance?

Hard (and informal) task one tries to mechanize

Modeling workflow

World

Big Data

- Literature
- Data bases
- Experiments

Representation

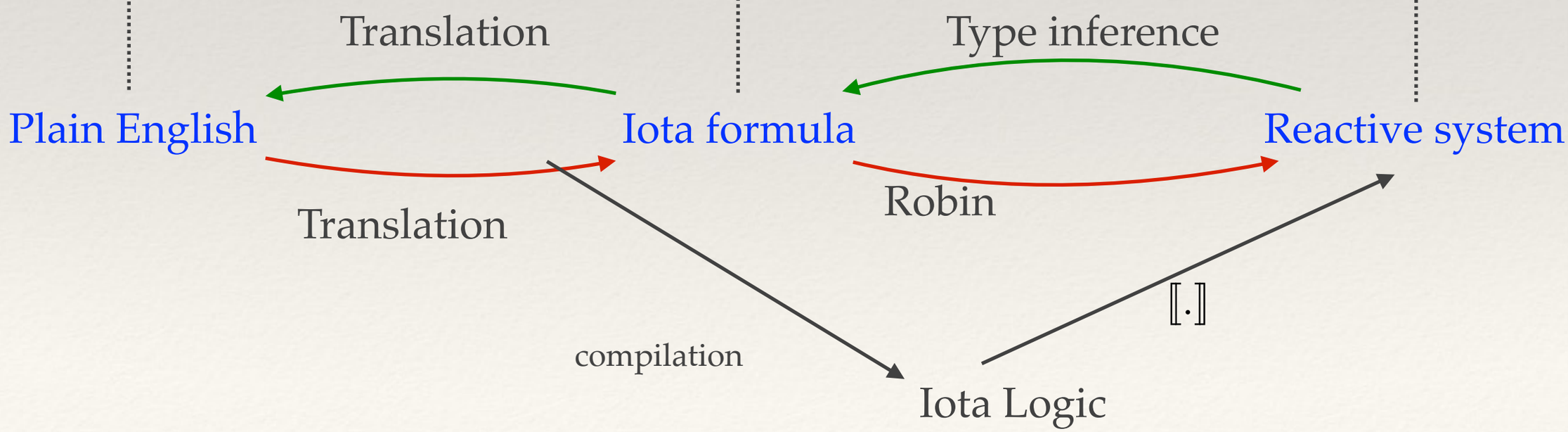
Knowledge

- Facts
- Assertions

Engineering

Model

- Facts
- Observables
- Initial state



Modeling workflow

World

Big Data

- Literature
- Data bases
- Experiments

Representation

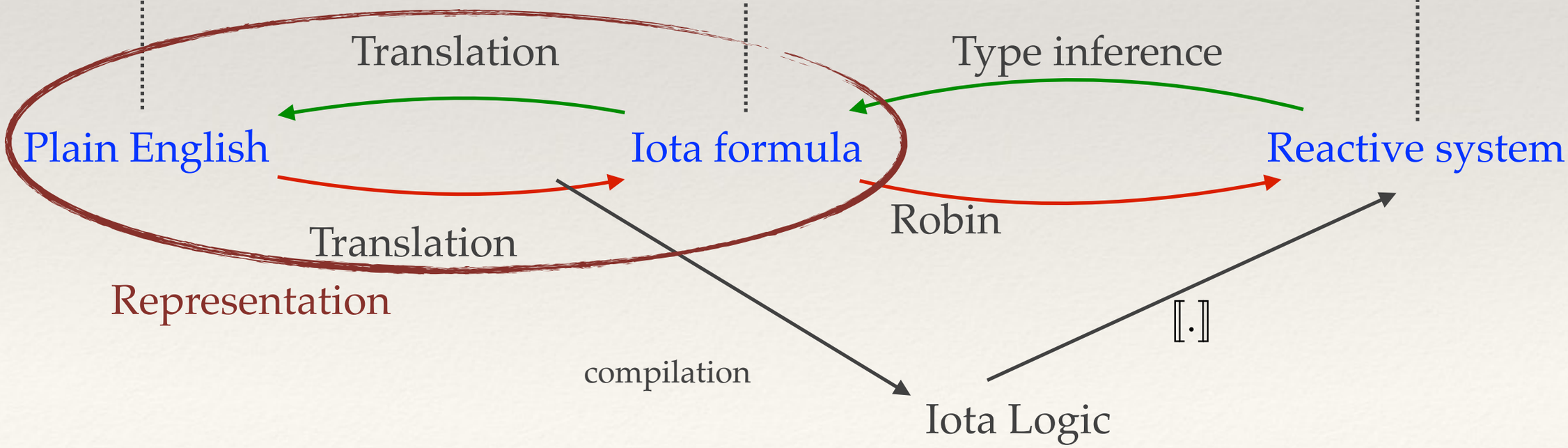
Knowledge

- Facts
- Assertions

Engineering

Model

- Facts
- Observables
- Initial state



Modeling workflow

World

Big Data

- Literature
- Data bases
- Experiments

Representation

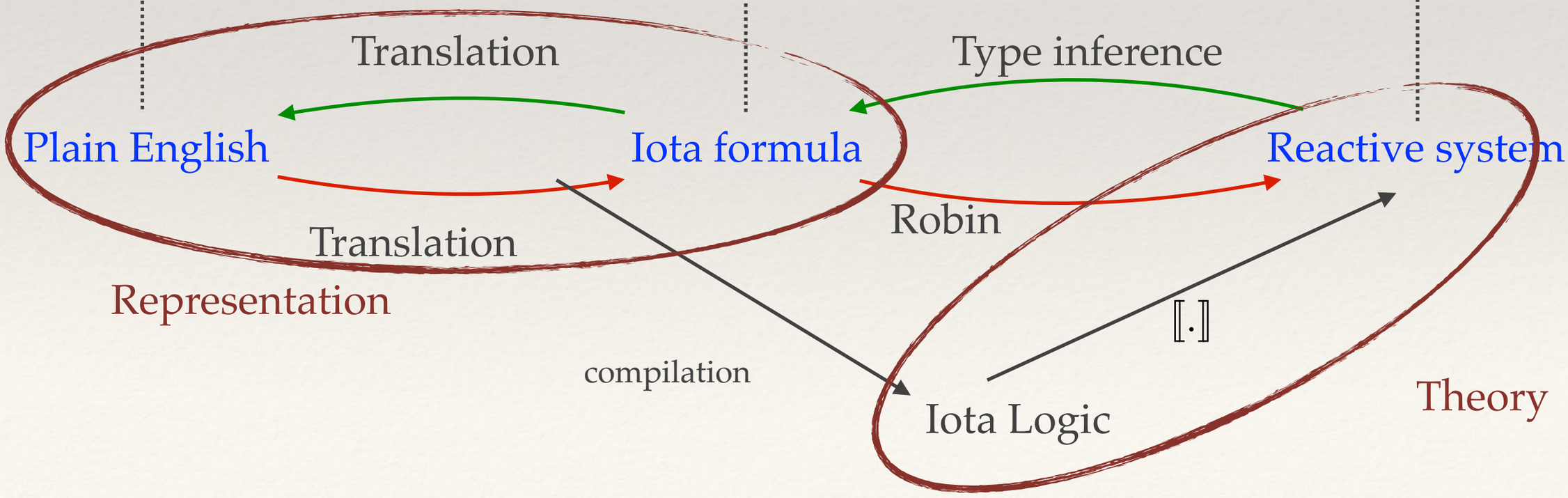
Knowledge

- Facts
- Assertions

Engineering

Model

- Facts
- Observables
- Initial state



Modeling workflow

World

Big Data

- Literature
- Data bases
- Experiments

Representation

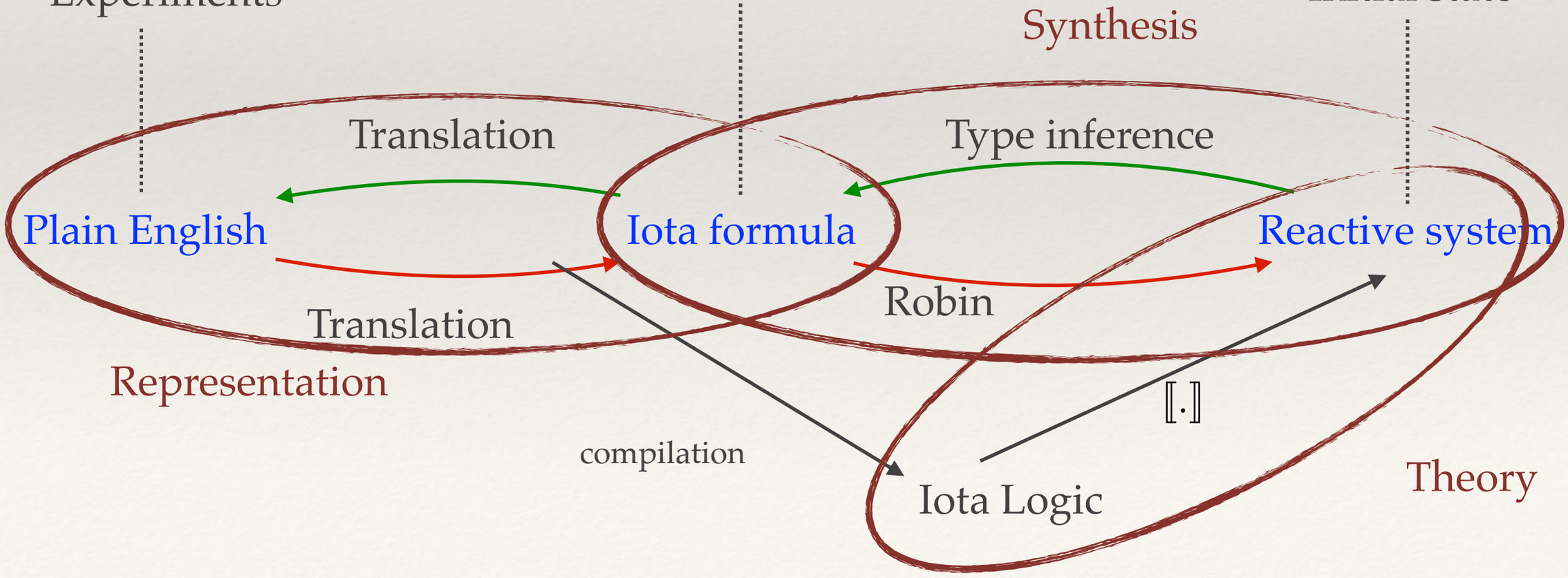
Knowledge

- Facts
- Assertions

Engineering

Model

- Facts
- Observables
- Initial state



Knowledge representation

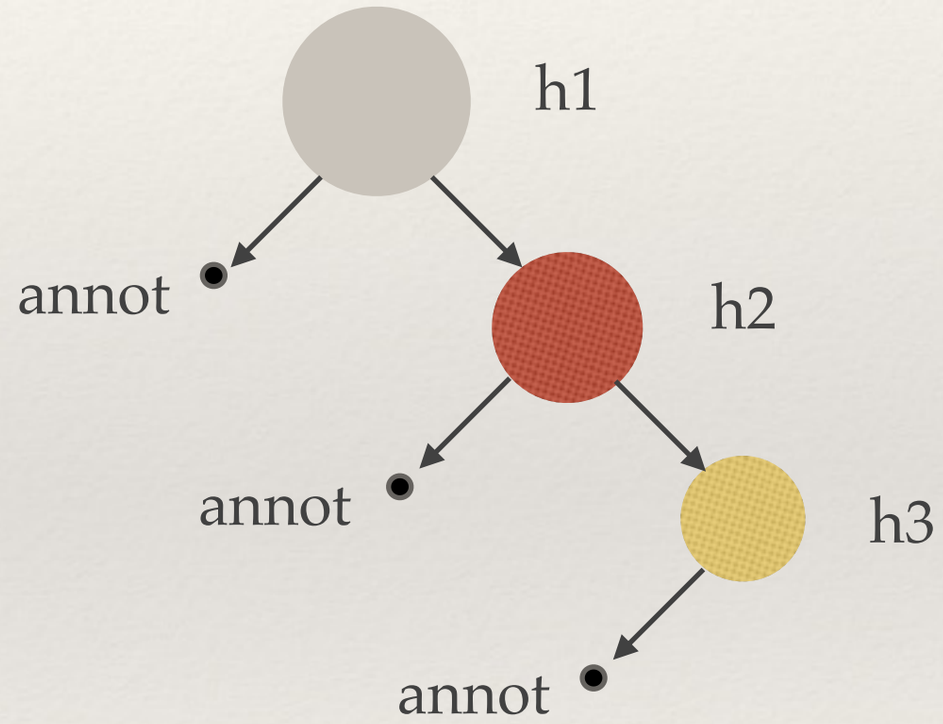
Robin

- (Eventually) an interactive modeling assistant
- Each statement can be compiled to a (fragment of) iota
- A Robin statement *t* is the type of a (possibly infinite) family *F* of *reactive systems*
- The aim of using Robin is to find a finite basis for *F* (e.g the smallest reactive system that has type *t)*

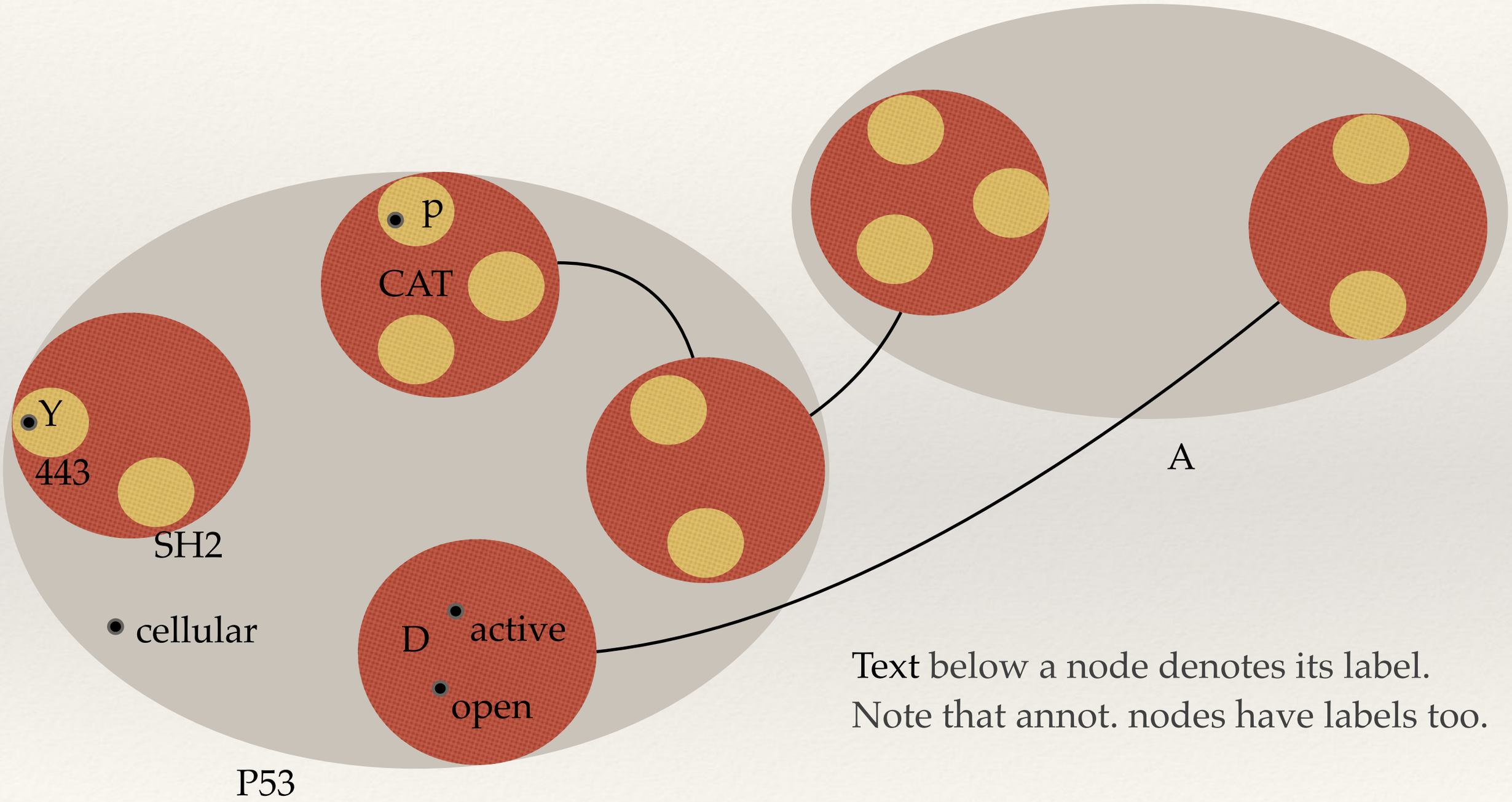
Statics

Robin's Structured graphs

- A restricted sort of structured graphs (to limit choice of encoding)
- Four kinds of nodes: *h1 (e.g proteins), h2 (e.g domains), h3 (e.g residuals)* and *flags*.
- Fixed labelling function for each node type (including *annot* nodes).
- Edges occur between *h2* nodes exclusively.



Representing molecules



Text below a node denotes its label.
Note that annot. nodes have labels too.

Notes on labels and annotations

- Nodes have are labelled by a small algebraic structure
- Typically a label l and its inverse $\neg l$
- One can also use (<,{MAPK,MAPK1,MAPK2,MAPK3}) with MAPK < MAPK*i* a partial order
- All annotation labels are constrained to have an inverse
- When a node is created, it comes with all its possible annotations and its default label

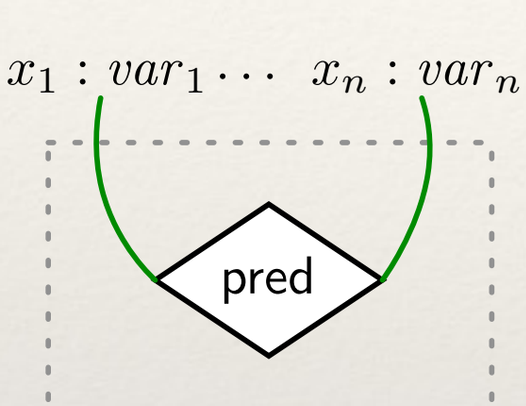
Atomic graph predicates

_ is_present	:	$var \to graph\_predicate$
_ has_domain _	:	$h_1 \to h_2 \to graph\_predicate$
_ has_residue _	:	$h_2 \to h_3 \to graph\_predicate$
_ has_annotation _	:	$h_1 + h_2 + h_3 \to annotation$ $\to graph\_predicate$
_ is_labelled _	:	$h_1 + h_2 + h_3 + annotation$ $\to label \to graph\_predicate$
_ is_bound_to _	:	$h_2 \to h_2 \to graph\_predicate$
_ = _	:	$var \to var \to graph\_predicate$

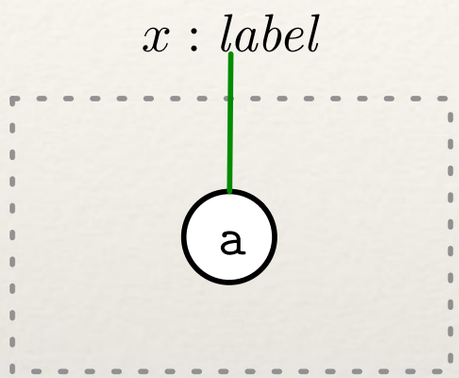
Graph predicates combinators

$$
\begin{array}{lcl}
G : graph\_predicate \ \mathbf{and} \ H : graph\_predicate & : & graph\_predicate \\
G : graph\_predicate \ \mathbf{or} \ H : graph\_predicate & : & graph\_predicate \\
\exists (x : var).G : graph\_predicate & : & graph\_predicate \\
\mathbf{not} \ G : graph\_predicate & : & graph\_predicate
\end{array}
$$

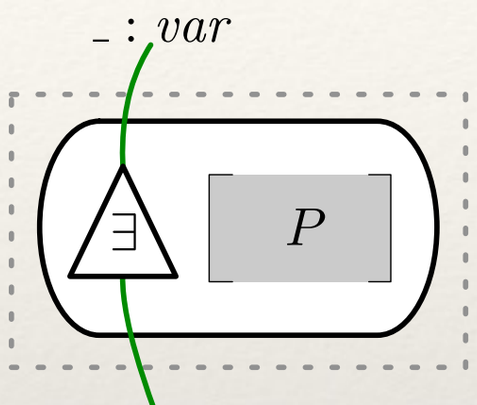
Graphical representation



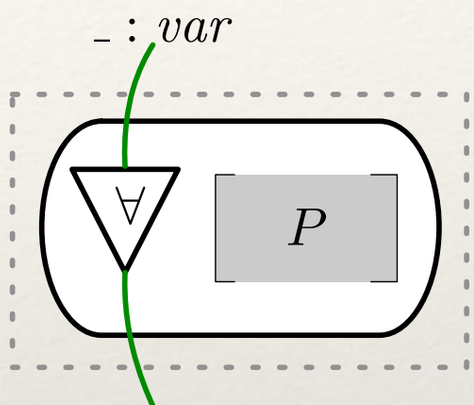
Atomic predicate



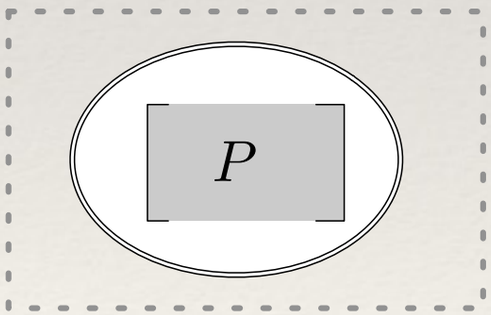
The label ‘a’



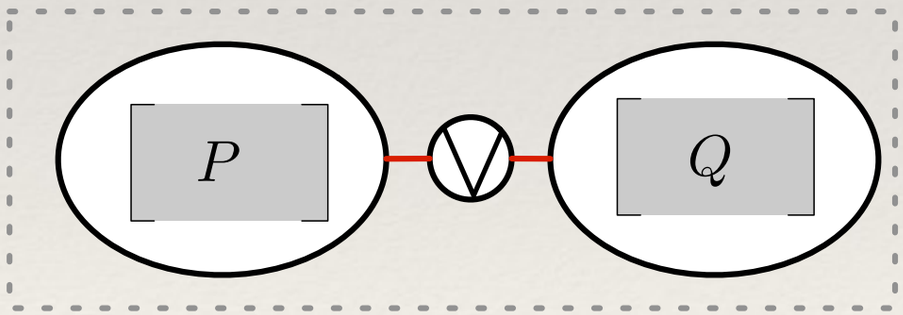
There exists x s.t P



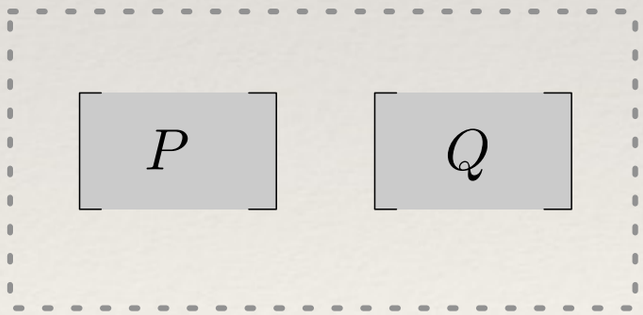
For all x s.t P



not P



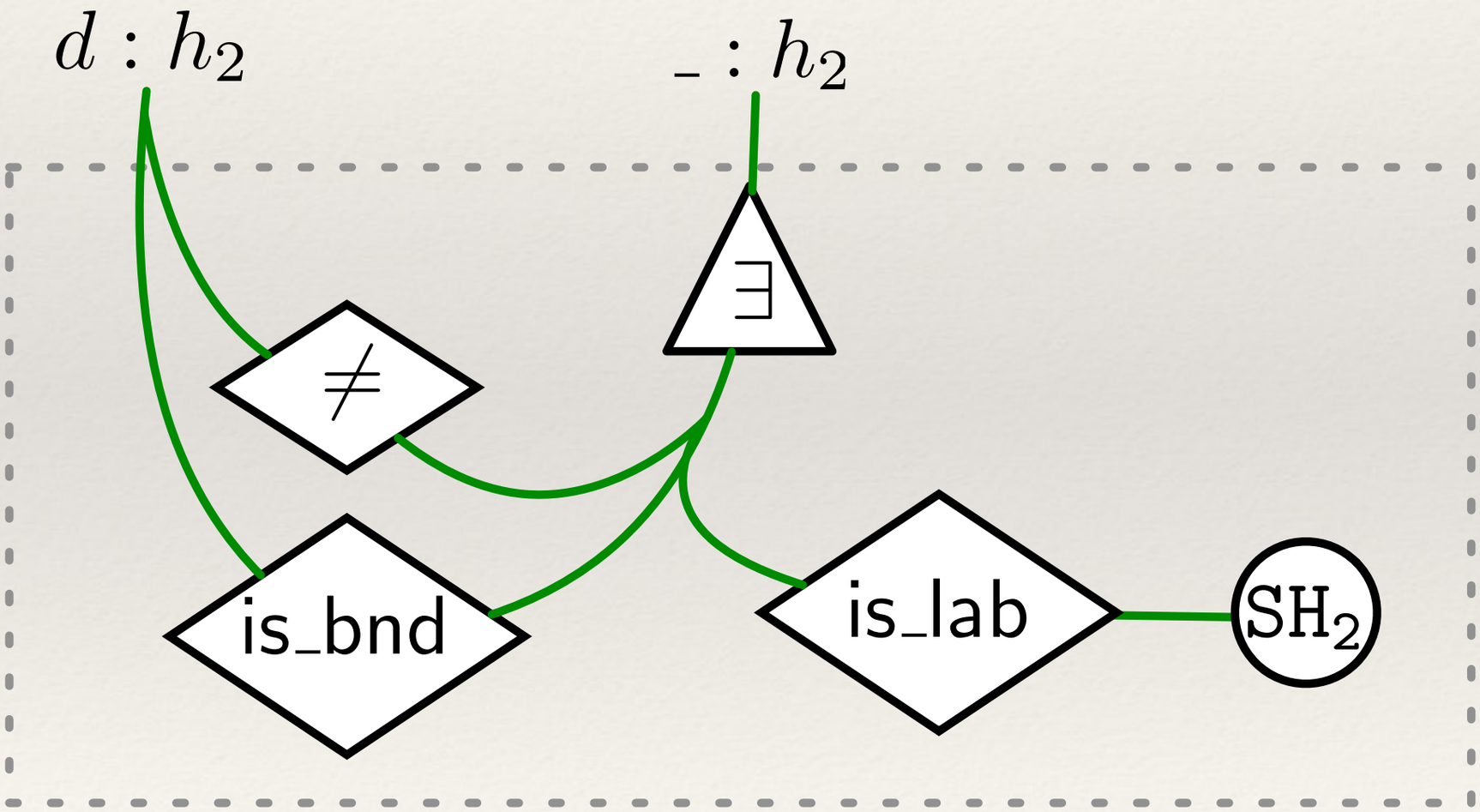
P **or** Q



P **and** Q

Types as mechanistic nuggets

$\exists d' : h_2 . d'$ is_labelled SH2 **and** d is_bound_to d' **and not** $d = d'$



A modeling language

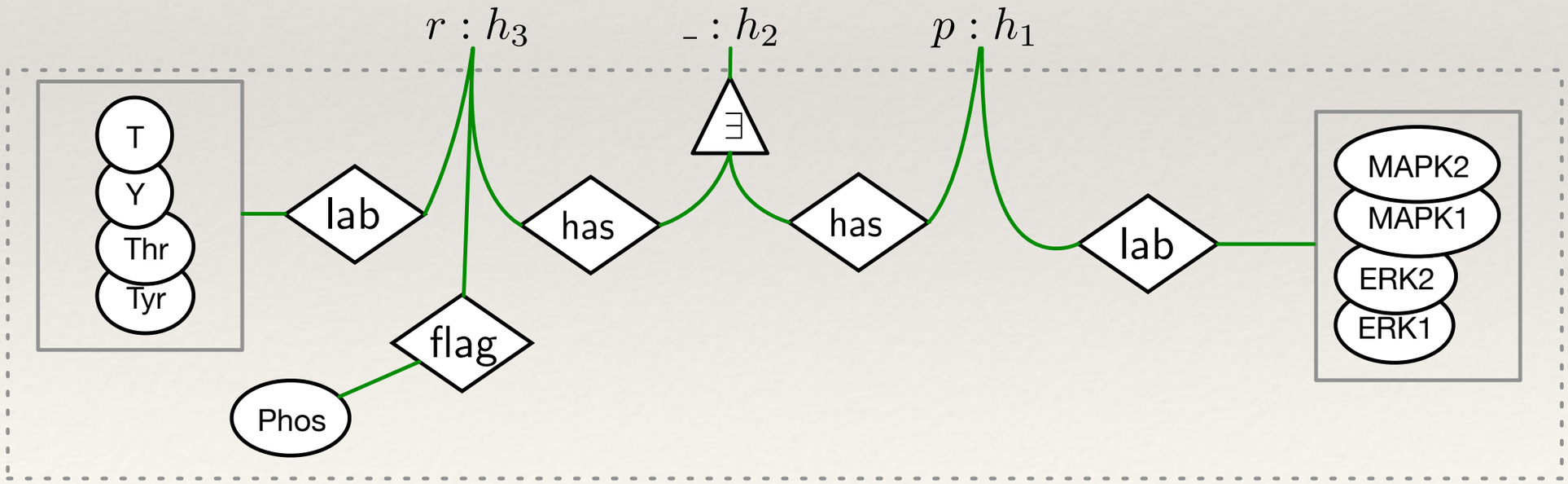
- Atomic predicates are building blocks to construct a modeling language (libraries)
- One uses variables and quantifiers to denote the unknown
- The structured graph interpretation is fixed to limit possible choices of encoding (e.g only *h2* nodes can bind)
- Libraries allows one to add flexibility.

Nugget factory

```
define active_ERK_at_residue(p:h1,r:h3) as
    p is_labelled 'MAPK'
    and
    Exists d:h2.
      p has_domain d
      and d has_residue r
      and r is_labelled ('Y' | 'T')
      and r has_annotation 'Phosphorylated'
```

Nugget factory

```
define active_ERK_at_residue(p:h1,r:h3) as
    p is_labelled 'MAPK'
    and
    Exists d:h2.
      p has_domain d
      and d has_residue r
      and r is_labelled ('Y' | 'T')
      and r has_annotation 'Phosphorylated'
```



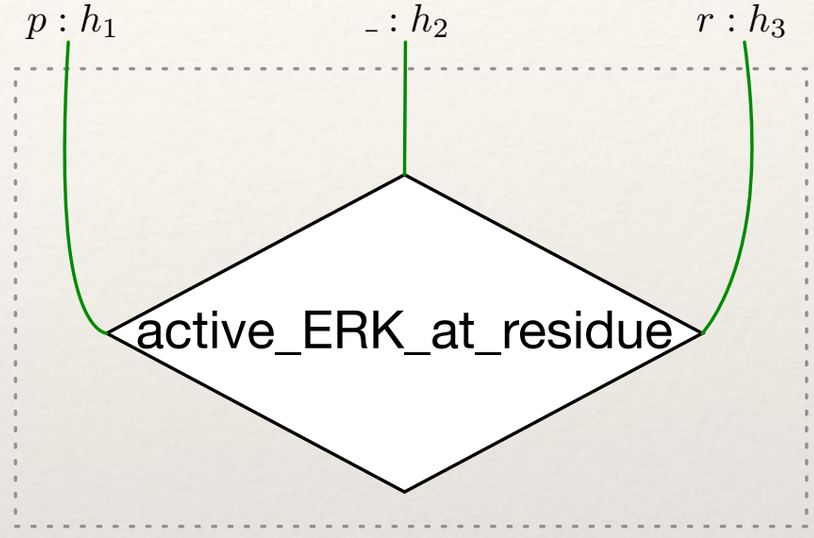
Nugget factory

```
define active_ERK_at_residue(p:h1,r:h3) as
    p is_labelled 'MAPK'
    and
    Exists d:h2.
      p has_domain d
      and d has_residue r
      and r is_labelled ('Y' | 'T')
      and r has_annotation 'Phosphorylated'
```

Nugget factory

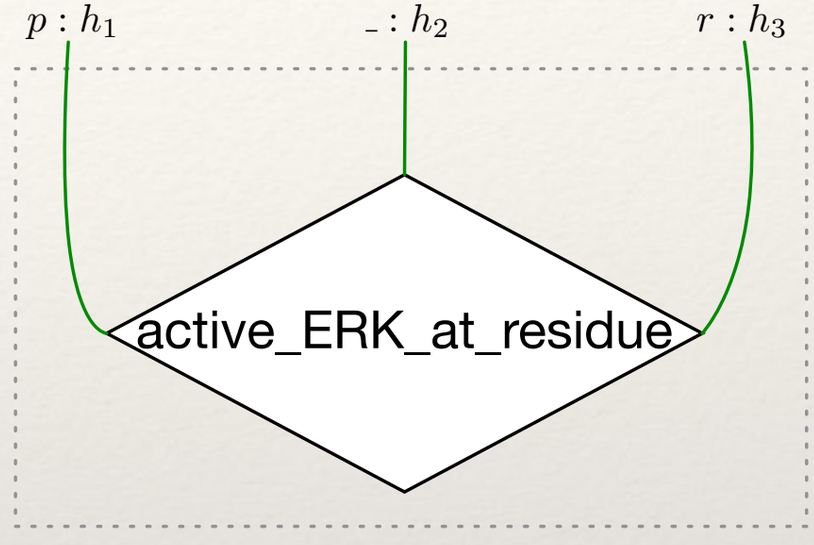
```
define active_ERK_at_residue(p:h1,r:h3) as
    p is_labelled ‘MAPK’
    and
    Exists d:h2.
      p has_domain d
      and d has_residue r
      and r is_labelled (‘Y’ | ’T’)
      and r has_annotation ‘Phosphorylated’

define active_ERK2(p:h1) as
    p is_labelled ‘ERK2’
    and
    Exists r:h3,s:h3 .
      and active_ERK_at_residue(p,r)
      and active_ERK_at_residue(p,s)
      and r is_labelled ‘Y’
      and s is_labelled ’T'
```

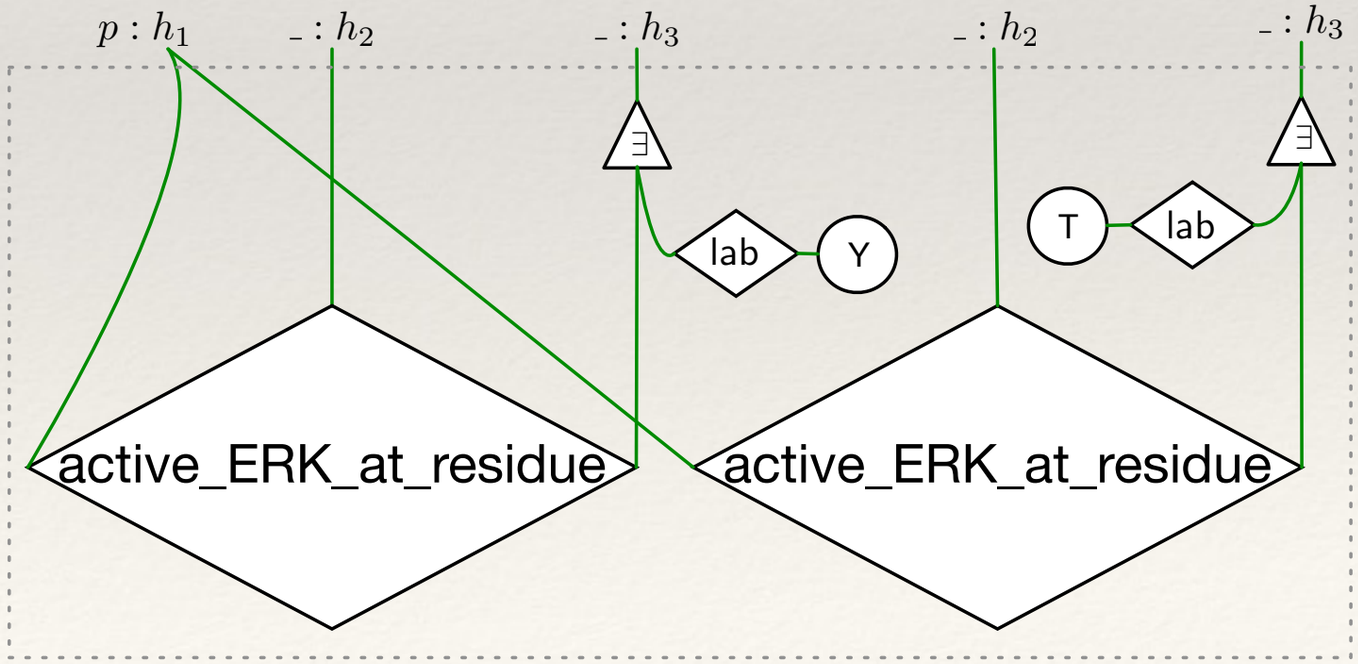


Nugget factory

```
define active_ERK_at_residue(p:h1,r:h3) as
    p is_labelled 'MAPK'
    and
    Exists d:h2.
      p has_domain d
      and d has_residue r
      and r is_labelled ('Y' | 'T')
      and r has_annotation 'Phosphorylated'
```



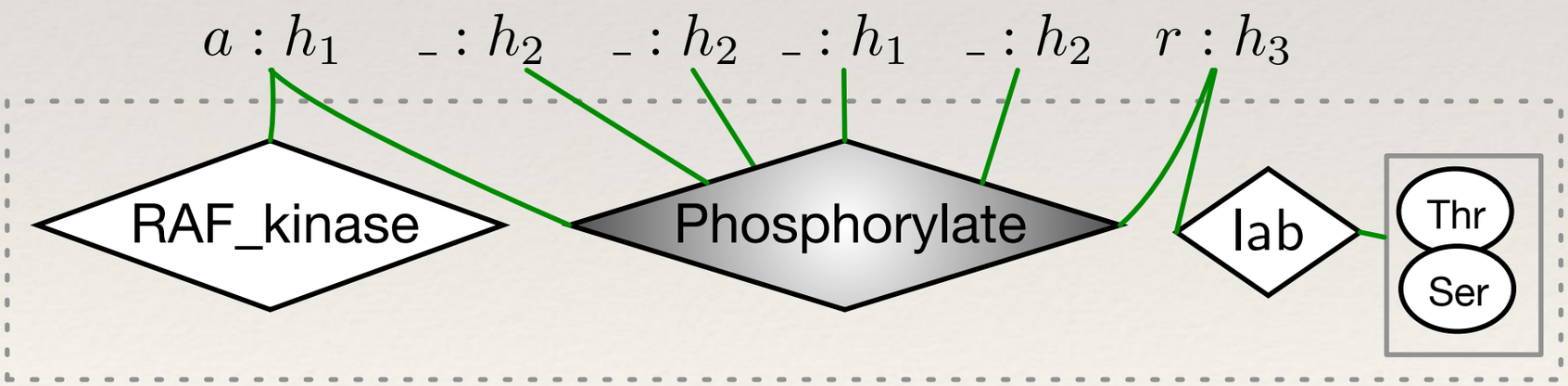

```
define active_ERK2(p:h1) as
    p is_labelled 'ERK2'
    and
    Exists r:h3,s:h3 .
      and active_ERK_at_residue(p,r)
      and active_ERK_at_residue(p,s)
      and r is_labelled 'Y'
      and s is_labelled 'T'
```



Action predicates

define RAF_kinase($a : h_1$) **as**
a is_labelled `A-RAF` **or** a is_labelled `B-RAF` **or** a is_labelled `C-RAF`

Can now be used as a building block for any rule that applies in general to RAF kinases.



« RAF-Kinases phosphorylate serine and threnonine residues »

Dynamics

Atomic action predicates

$$
\begin{array}{lcl}
\text{bnd}^{\star} & : & h_2 \rightarrow h_2 \rightarrow action\_predicate \\
\text{brk}^{\star} & : & h_2 \rightarrow h_2 \rightarrow action\_predicate \\
\text{upd}^{\star} & : & annotation \rightarrow label \rightarrow action\_predicate \\
\text{add}^{\star} & : & var \rightarrow label \rightarrow action\_predicate \\
\text{del}^{\star} & : & var \rightarrow action\_predicate
\end{array}
$$

Reactive systems combinators

$$
\begin{array}{lcl}
A : action\_predicate \;\&\; B : action\_predicate & : & action\_predicate \\
A : action\_predicate \;;\; B : action\_predicate & : & action\_predicate \\
A : action\_predicate \;\mathbf{when}\; G : graph\_predicate & : & action\_predicate \\
A : action\_predicate \;\mathbf{alternatively}\; B : action\_predicate & : & action\_predicate \\
\exists (x : var).(A : action\_predicate) & : & action\_predicate
\end{array}
$$

Non native predicate constructor:

$$\mathbf{define}\ \mathsf{f\_name}(\tilde{x})\ \mathbf{as}\ (Q : predicate)[\mathbf{in}\ P : predicate]$$

with $fv(Q) = \tilde{x}$ and $S \in fv(P)$.

Protein level interactions

define Agt.bind*$(x : h_1, y : h_1)$ **as**
$\exists d, d' : h_2 .$
bnd* d d' **when**
and x has_domain d
and y has_domain d'

« Protein [x] binds protein [y] »

define Agt.has_mod$(x : h_1, y : h_3, z : label)$ **as**
$\exists m : annotation . \exists p : h_2 .$
x has_domain p
and p has_residue y
and y has_annotation m
and m is_labelled z

« Protein [x] has modification [z] at residue [y] »

in
define Agt.is_uphos$(x : h_1, y : h_3)$ **as** Agt.has_mod$(x, y, \neg$Phos$)$

« Protein [x] is Unphosphorylated at residue [y] »

Modeling with a biological culture

What is activation?

*« [a] **activates** [b] »*

define Var.Toggle$^\star(x : var, y : label)$ **as**
$\exists m.(\text{upd}^\star\ m\ y$ **when** x has_annotation m **and** m is_labelled $\neg y)$
in
define Agt.Is_bound$(x : h_1, y : h_1)$ **as**
$\exists d, d'.(d$ is_bound_to d' **and** x has_domain d **and** y has_domain $d')$
in
define Agt.Has_residue$(x : h_1, y : h_3)$ **as**
$\exists d.(x$ has_domain d **and** d has_residue $y)$
in
$\exists r, mod.($
Var.Toggle$^\star(r, mod)$ & Var.Toggle$^\star(b, \texttt{active})$ **when**
Agt.Is_bound(a, b) **and** Agt.Has_residue(a, r)
$)$

Evaluating a new nugget

World	*Representation*	*Engineering*	*Compilation*
Big Data	Knowledge	Model	Reactive System
Literature	Iota Ψ	Iota Φ	Kappa
	Generic facts	Specific facts	

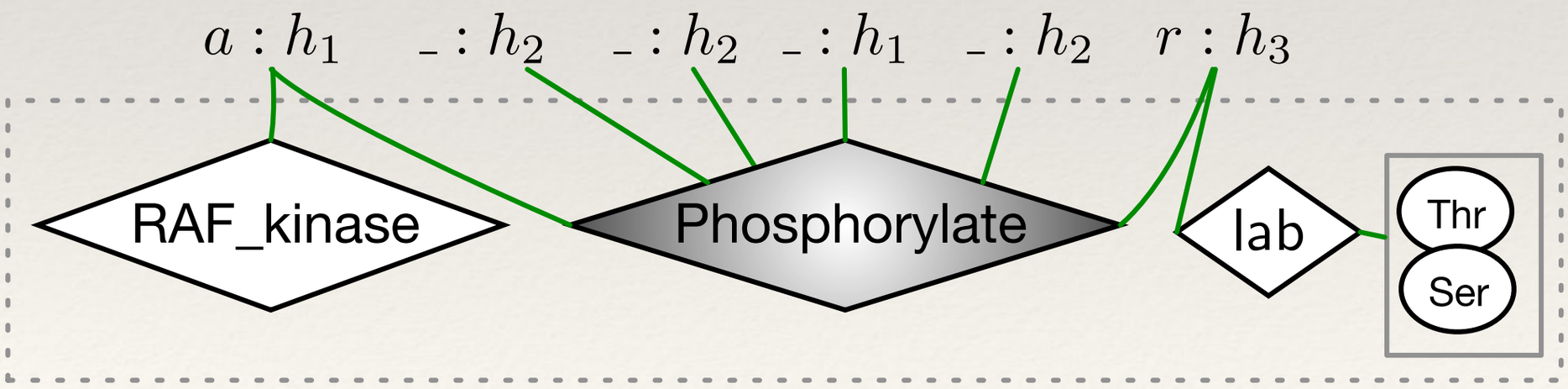
A new fact ϕ can be:

- strong $\exists \psi \in \Psi . \phi \implies \psi$
- grounded $\exists \psi \in \Psi . \phi \iff \psi$
- weak $\exists \psi \in \Psi . \psi \implies \phi$
- Assumed otherwise

These definitions are modular in the definition of $\psi \in \Psi$

Generic facts

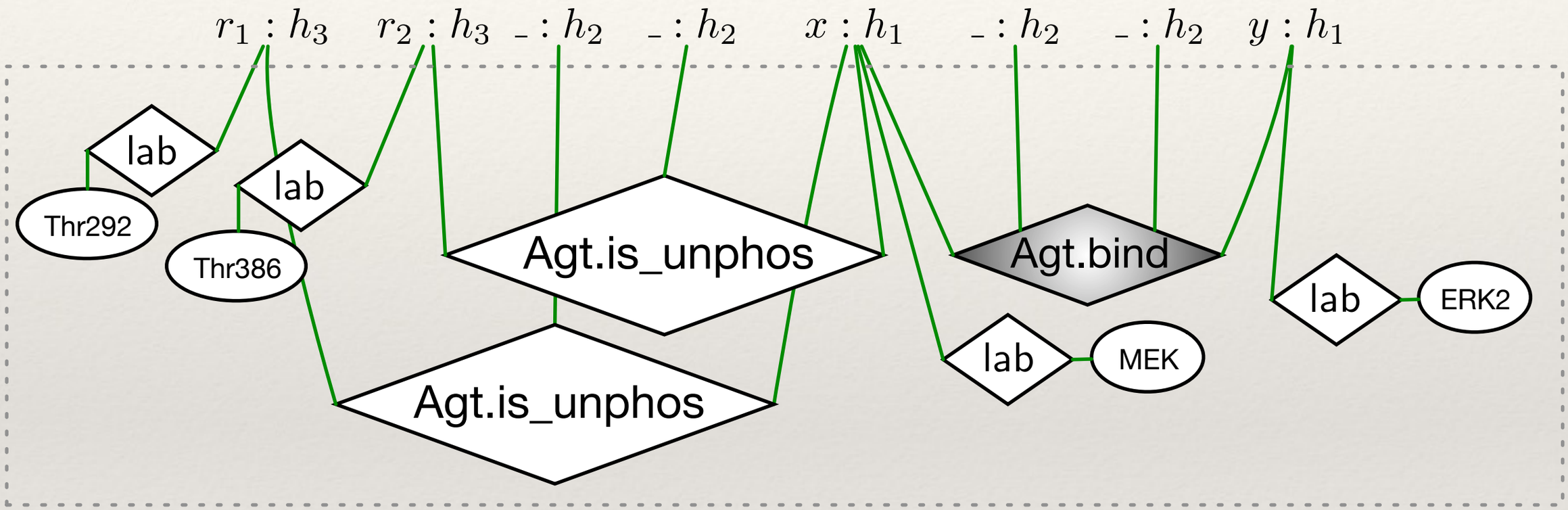
```
define Phosphorylate* (a:h1, r:h3) as
    Exists b:h1,s,t:h2.
        Agt.bind* a s b t when
          Agt.has_residue b r
          and r has_annotation  ¬'Phos' ;
Var.Toggle* r 'Phos' when
    Agt.is_bound a s b t
    and Agt.has_residue b r
```



« RAF-Kinases phosphorylate serine and threnonine residues »

A grounded new nugget

« ERK2 binds unphosphorylated MEK at residues Thr292, Thr386 »




```
Agt.bind* x y when
    x is_labelled ERK2 and y is_labelled MEK
    and Agt.is_unphos y r1
    and Agt.is_unphos y r2
    and r1 is_labelled 'Thr292'
    and r2 is_labelled 'Thr386'
```

Biological invariants

$$
\begin{array}{lcl}
G : graph\_predicate & : & invariant \\
(G : graph\_predicate)^{\star} & : & invariant \\
\neg(I : invariant) & : & invariant \\
\exists(x : var).(I : invariant) & : & invariant \\
\forall(x : var).(I : invariant) & : & invariant \\
(I : invariant) \wedge (J : invariant) & : & invariant \\
(I : invariant) \vee (J : invariant) & : & invariant \\
 & & \\
\textbf{weak\_assert}\ (I : invariant) & : & assertion \\
\textbf{assert}\ (I : invariant) & : & assertion \\
\textbf{ensure}\ (I : invariant) & : & assertion
\end{array}
$$

We use the usual derived operators:

$$
\begin{array}{lll}
I \Rightarrow J & =_{def} & \neg I \vee J \\
I \iff J & =_{def} & (I \Rightarrow J) \wedge (J \Rightarrow I)
\end{array}
$$

Sculpting the model

« A can only bind another protein on its open SH2 domain »

assert
Forall a:h1,d:h2,b:h1,d':h2.
(Agt.bound a d b d' **or** (Agt.bound a d b d')*
and a is_labelled 'A')
iff
(d is_labelled 'SH2'
and d has_annotation 'open')

Dumps interpretations that do not satisfy the assertion

« A is active when it is phosphorylated »

ensure
Forall a:h1.
Agt.is_phosphorylated a **and** a is_labelled 'A'
iff a has_annotation 'active'

Replaces interpretations that do not satisfy the assertion with a more precise graph that does (if any)

Synthesis/Realization

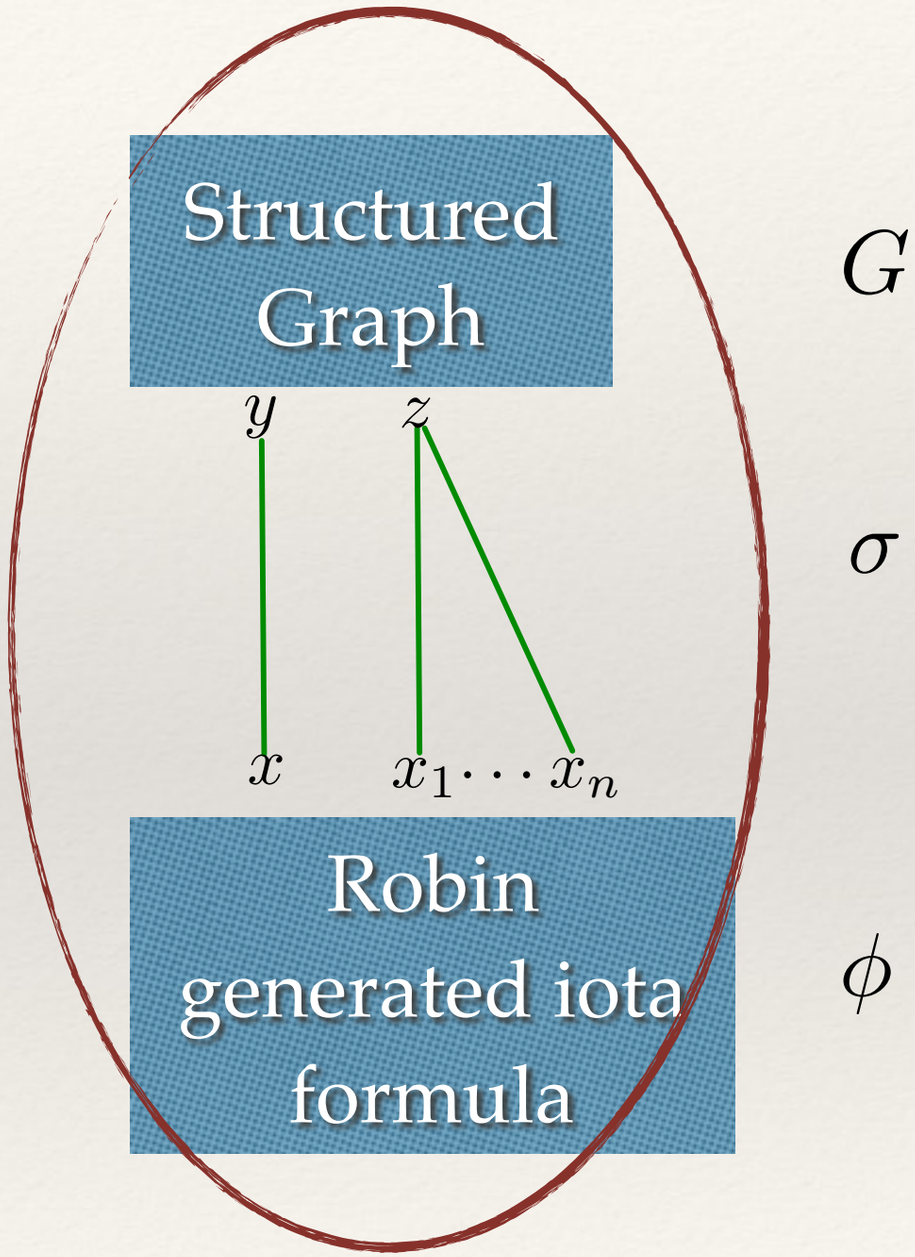
(sketch)

Generating a basis

- Recall a formula denotes a set of reactive systems (a structured graph equipped with actions) which are models of the formula.
- For a formula F, one wishes to find a canonical set of structured graphs such that all other graphs that model F are more refined that one of its elements

Assignation

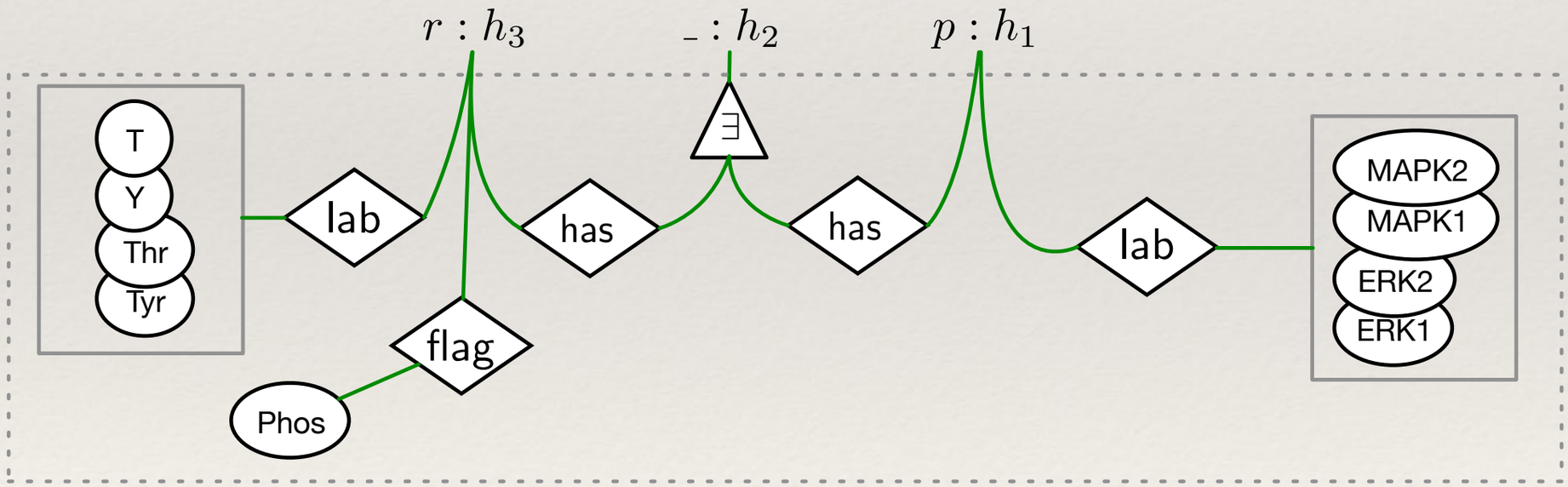
- Type inference (**input**: a structured graph, **output**: a iota formula)
- Realization (**input**: a iota formula, **output**: typed graph)
- Type checking (**input**: an annotated structured graph, **output**: typed graph)



Reactive system

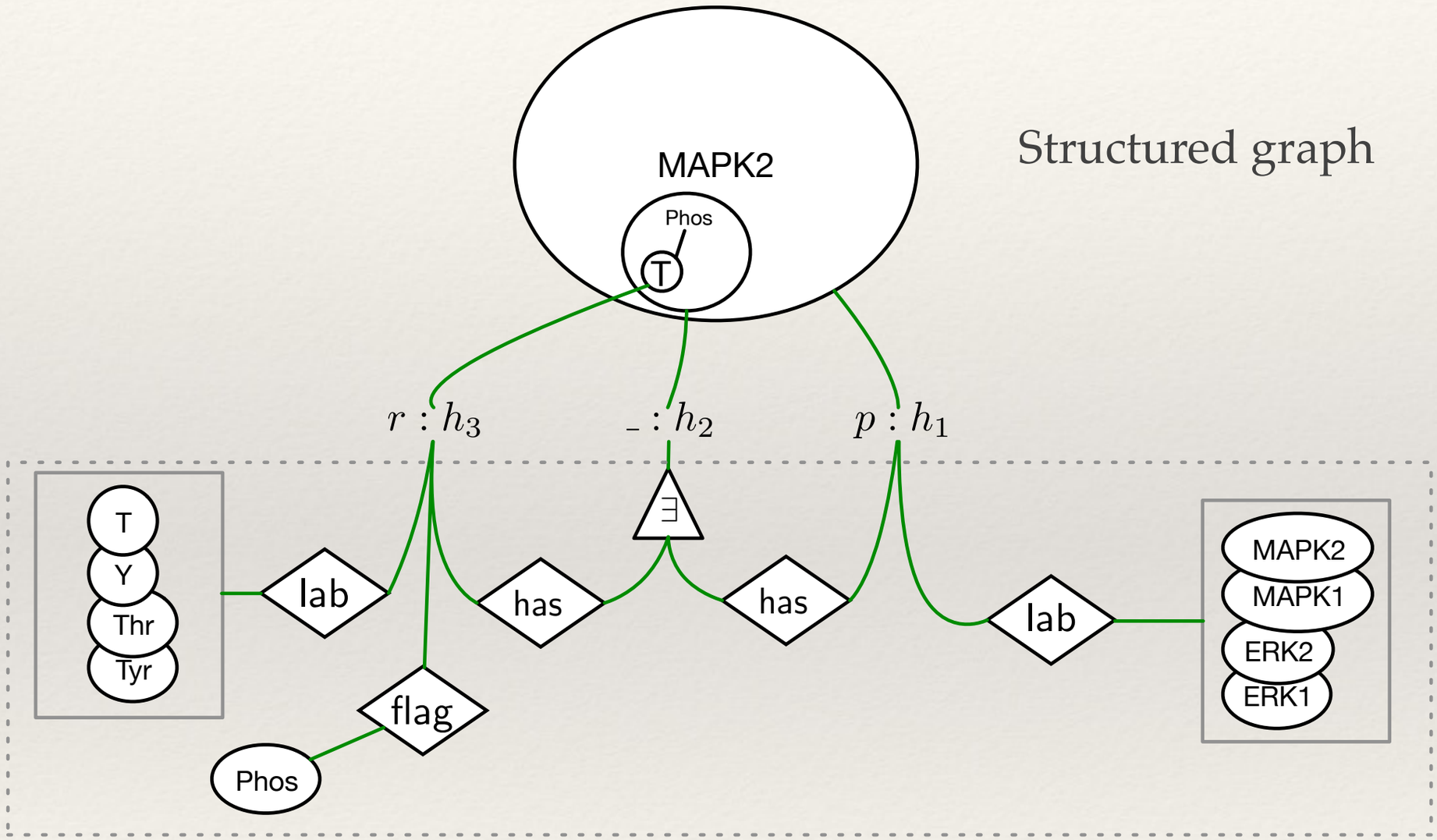
Realization (single basis)

Structured graph



Iota formula

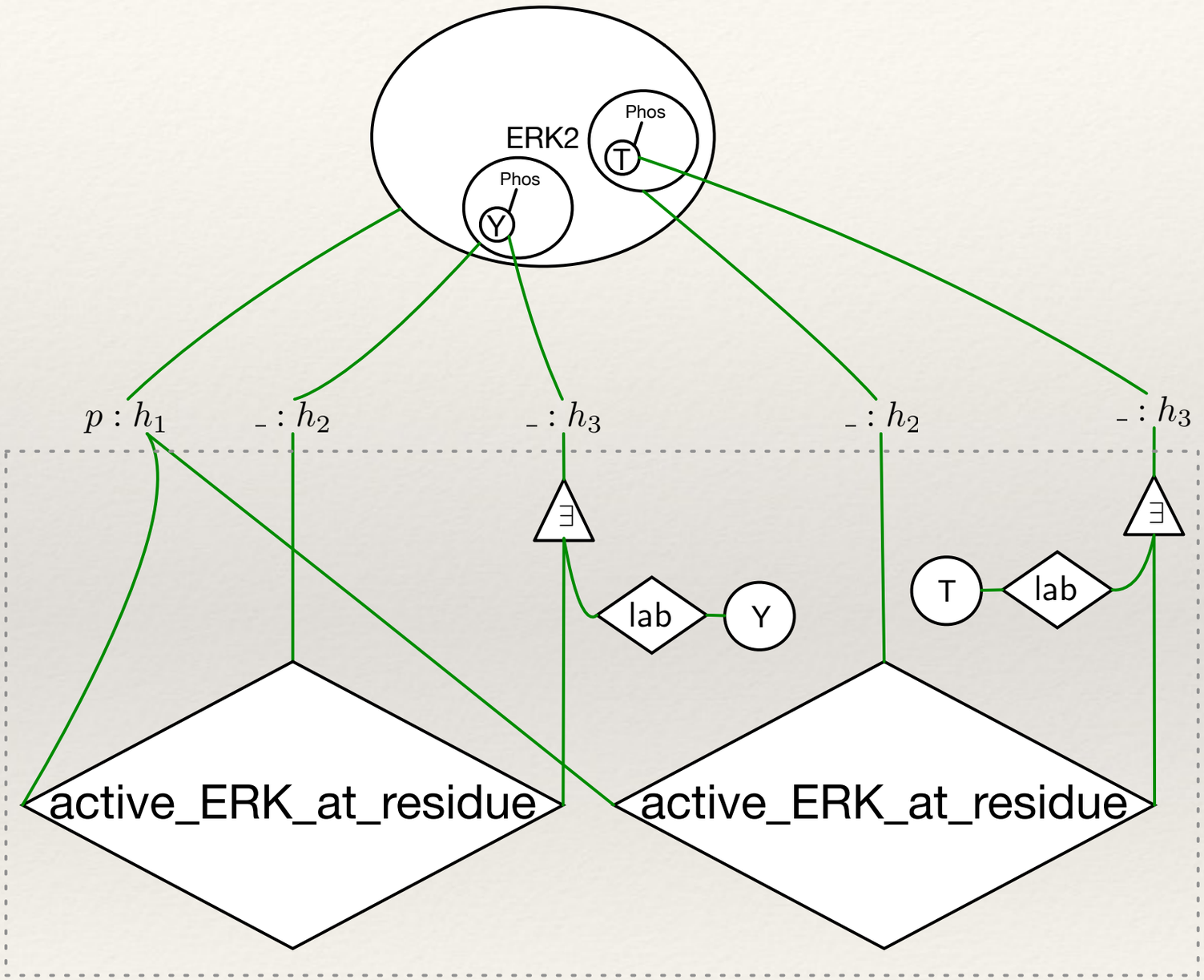
Realization (single basis)



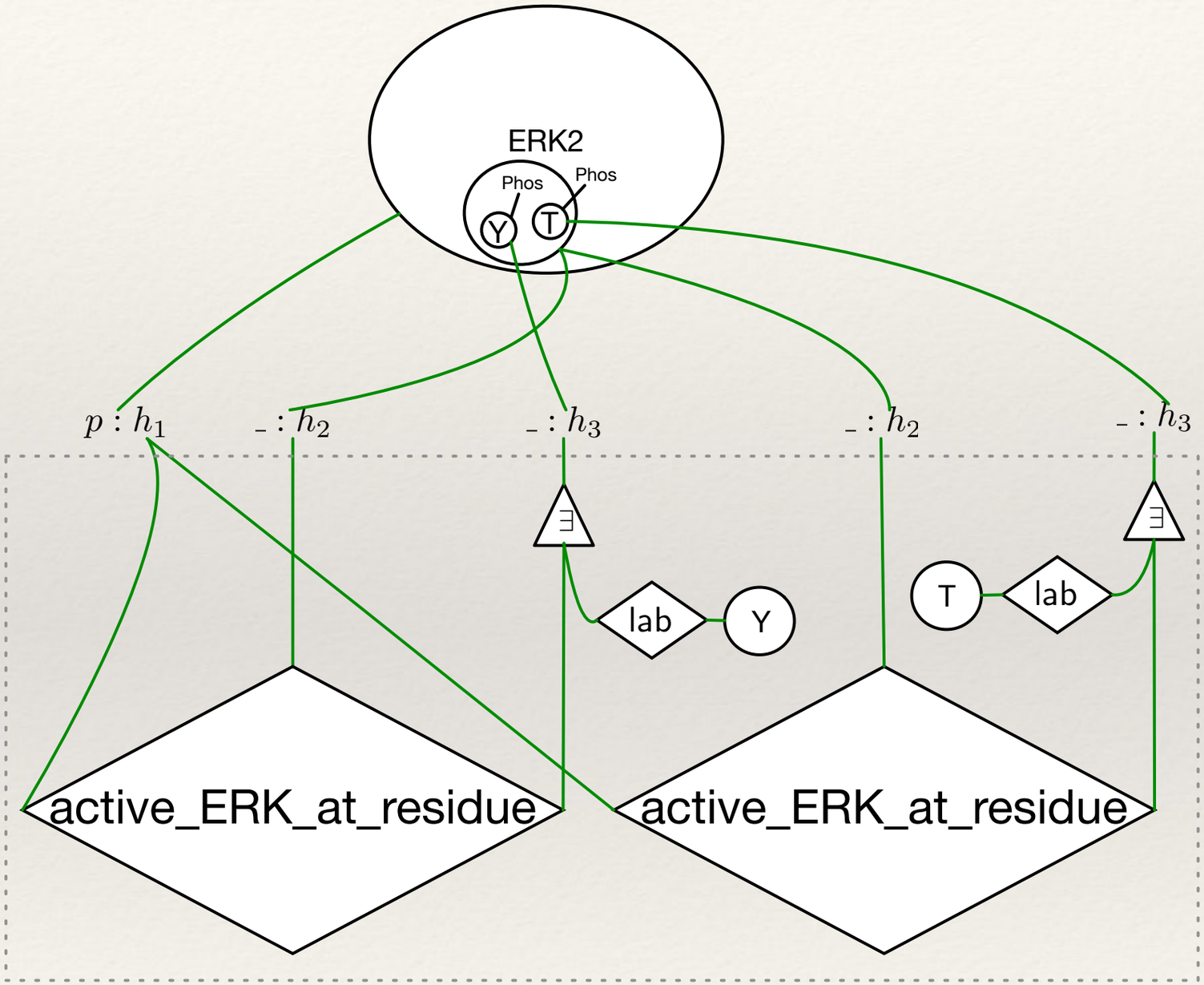
Structured graph

Iota formula

Realization (multiple basis)

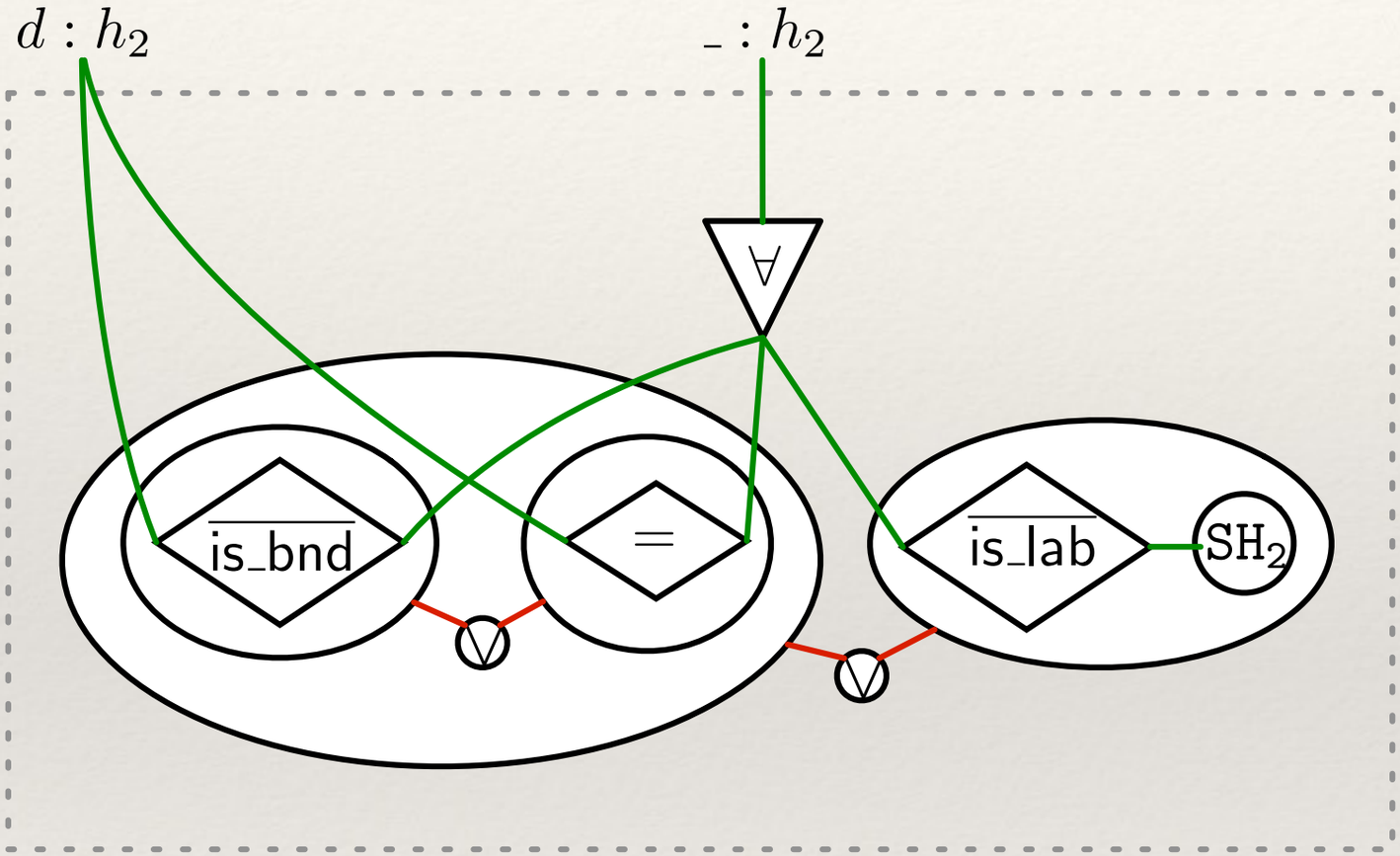


Realization (multiple basis)



Realization (no finite basis)

« [d] is not bound to any SH2 domain »



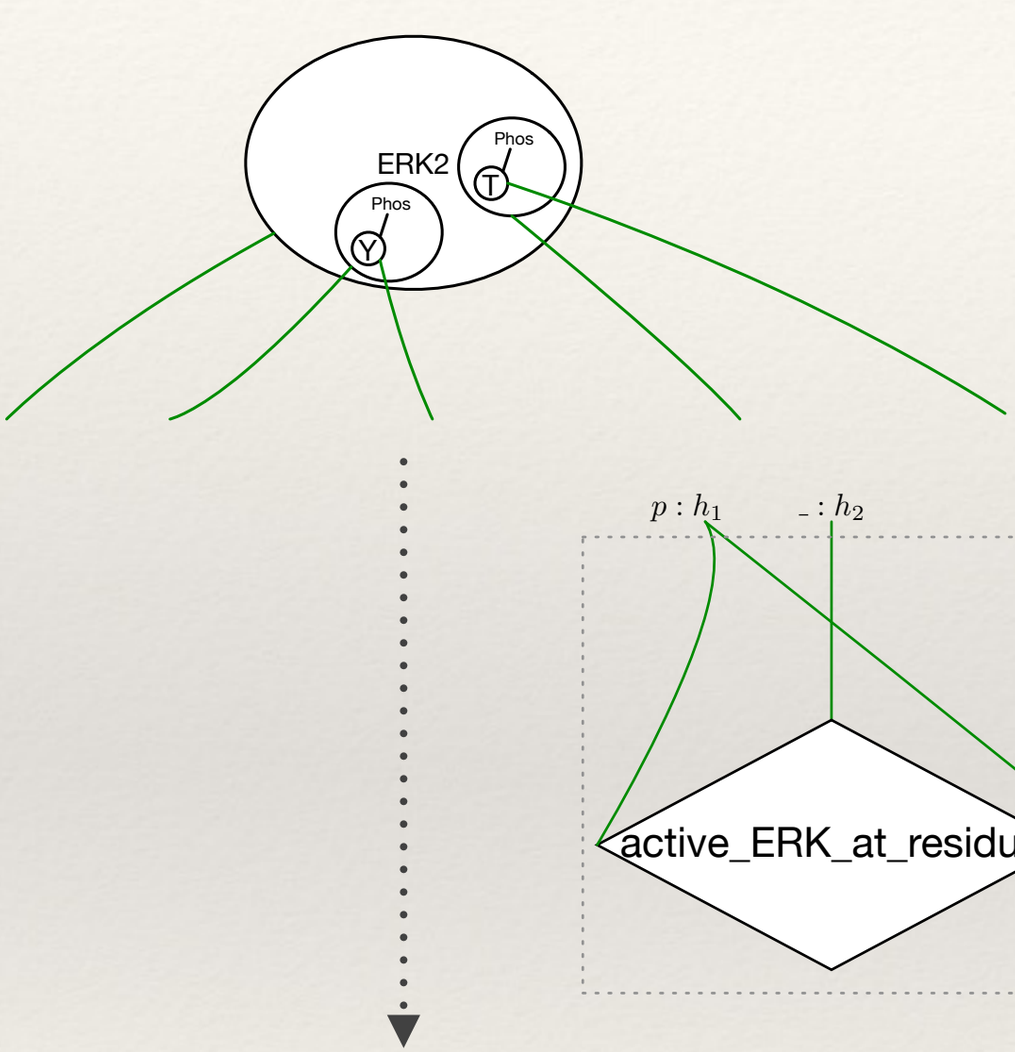
ensure

Forall d,d':h2.
d is_labelled 'SH2'
and d is_bound_to d'
iff
d' has_annotation 'busy'

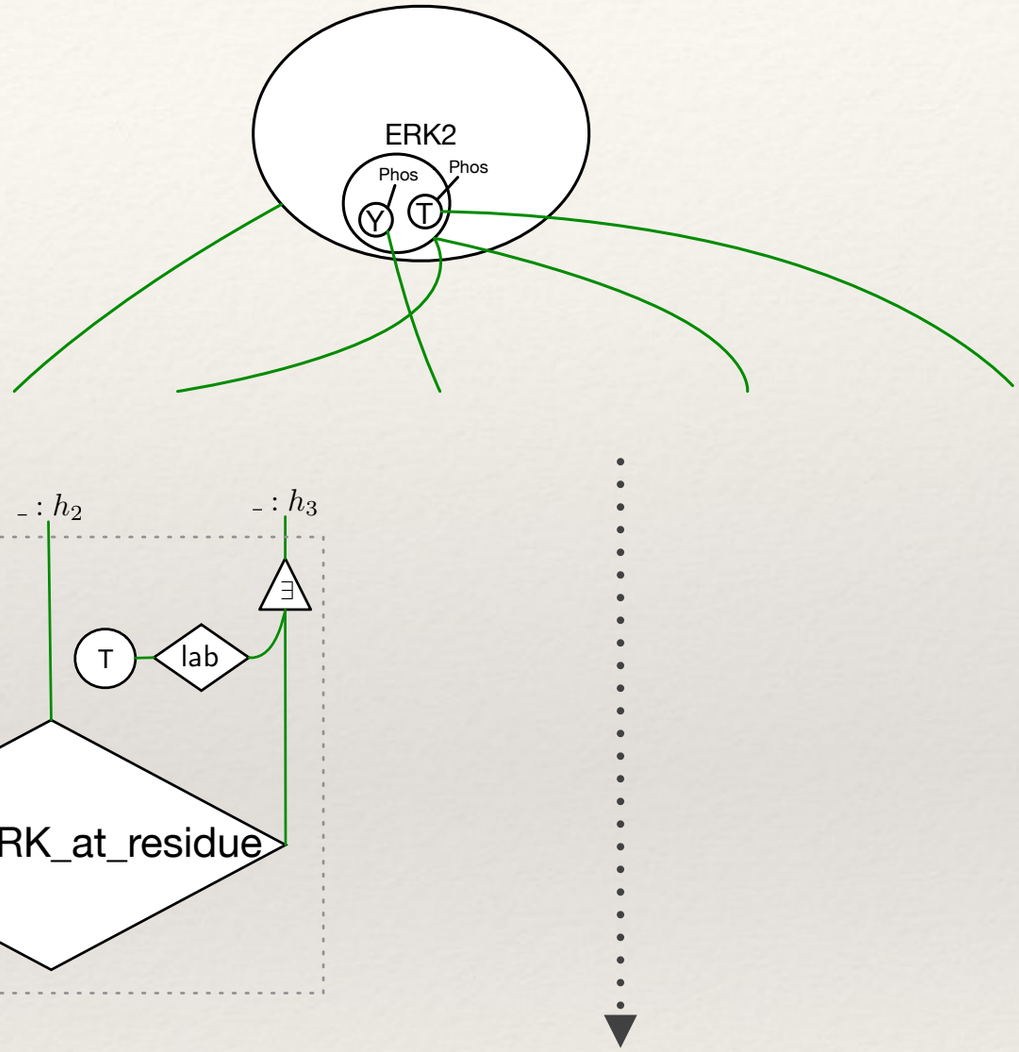
This type not denoted by a finite basis of reactive systems.
However one can rely on invariants to interpret it in a finite way. This is a **compilation** issue, not a KR issue.

It suffices then to ask that d' has the annotation ¬'busy'

Notes on compilation to Kappa



OR



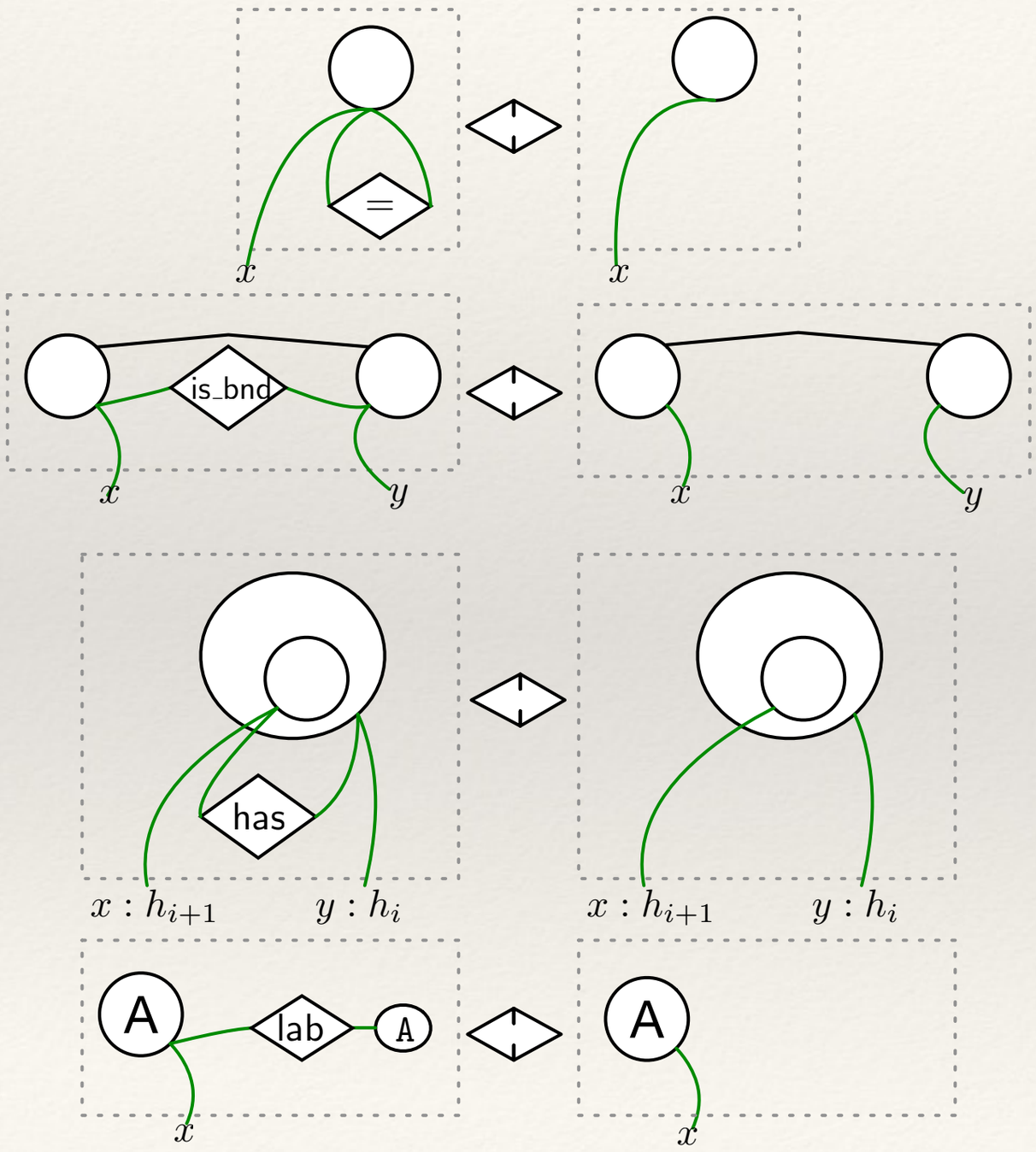
ERK2(d1*!1*,d2*!2*),**ERKD_1**(p*!1*,Y*~phos*),**ERK_D2**(p*!2*,T*~phos*)

ERK2(d1*!1*),**ERK_D**(p*!1*,Y*~phos*,T*~phos*)

If no nugget mentions ERK2 domains, both compilations will behave identically

Typed graph operational semantics (sketch)

Reducing types



Suppose $G \circ \phi \rightarrow^* G \circ \psi$

Then

$\forall G' > G.\ (G \models \phi \iff G' \models \psi)$

So for any graph G′ in which G embeds, the effect of ϕ can be applied provided G′ models ψ

In particular $G \circ \phi \rightarrow^* G \circ \alpha$ induces a context free reactive system (actions are unconditionally applied).

Also $G \circ \phi \rightarrow^* G \circ \bot$ denotes a type mismatch (the effect of ϕ cannot be applied)

Modeling

- Iota (Program type): *a logical formalism for the KR.*
- Robin (Programming language): *a modeling assistant on can teach biology to.*
- Kappa (Machine code): *a special kind of (context free) reactive system.*
- A model (A proof): a iota formula the assembly of which is grounded by Robin